Programa de Pós-graduação em Sistemas de Informação
Escola de Artes, Ciências e Humanidades
Universidade de São Paulo

SIN5022 - Teste de Software

Prof. Marcelo Medeiros Eler

PROJETO 01 - MÁQUINAS DE ESTADOS

Mateus Miranda de Frias
Número USP: 8516837

Objetivo do projeto

O objetivo do projeto é criar um gerador automático de testes usando o modelo de uma máquina de estados finitos. A partir de duas entradas (um arquivo CSV com a tabela de transição que cruza os estados com os eventos, e um mapeamento de eventos e ações da máquina - métodos que representam os eventos em si implementados, que levam o sistema de um estado a outro.

Linguagem e requisitos para execução

O projeto foi desenvolvido em JAVA, usando o projeto base de leitor de arquivos CSV disponibilizado pelo Professor. Todo o código se encontra no arquivo *main.java*.

Para executar o projeto, basta executar a classe *main.java*. Todas as saídas serão impressas no console. Os resultados do projeto serão apresentados com as entradas conforme oferecidas na descrição do mesmo, mas o sistema está pronto para qualquer entrada caso as regras abaixo sejam seguidas.

O sistema procura por dois arquivos CSV na raiz do projeto, *eventos.csv* e *transicao.csv*. O arquivo *eventos* precisa separar usar o separador de colunas “;”, pois as chamadas de métodos podem possuir vírgulas. Já o arquivo *transicao* deve possuir “,” como separador de colunas. Em ambos os arquivos, todos os espaços desnecessários devem ser removidos, e o cabeçalho deve ser mantido como existente nos arquivos.

Implementação

No geral, a implementação entrega os objetivos do projeto mas sem atingir bons padrões de estrutura de código ou eficiência, pois foi meu retorno a programação de um grande projeto depois de alguns anos cumprindo funções menos técnicas no mercado de trabalho. Alternei o uso de matrizes, vetores e listas de String, e centralizei quase toda a execução no próprio método *main* da classe principal.

Após ler e armazenar os dois arquivos CSV, extraio os eventos e os estados das primeiras colunas. Monto a Árvore de Transição adicionando sufixos nos estados, com o contador de vezes que se repetem. Também identifico e salvo os Nós Folha da árvore, pois ajudarão a identificar e montar todos os caminhos possíveis da árvore.

Para identificar os caminhos possíveis, a partir da raiz da árvore, cruzo todas as arestas das árvores que começam com um estado, com aquelas que terminam com o mesmo estado, atualizando a lista de caminhos a cada aresta analisada para que o resultado final sejam caminhos realmente válidos, ou seja, que se iniciam no

estado inicial do sistema (raiz da árvore) e se encerram em um estado final (nó folha).

Para gerar os testes a partir dos caminhos possíveis, gero uma matriz de transições, com as colunas Estado Inicial, Evento e Estado Final, e busco os eventos e estados que o script de teste deve usar para cada caminho possível encontrado.
Ao acessar o primeiro estado de um caminho, imprimo o primeiro *assertEquals*, que faz a validação do estado inicial do sistema. A partir disso, cada estado no caminho gera um par de instruções: uma chamada do método relacionado ao Evento (vindo do mapeamento no CSV *eventos*) e mais uma verificação de Estado com *assertEquals*.

Saídas

Usando as entradas conforme oferecidas na descrição do projeto, e com as modificações e regras descritas na seção “Linguagem e requisitos para execução”, os arquivos de entrada ficaram nessa configuração:

eventos.csv:
system proxy;SDevice
home;SDevice.home()
wrong password;SDevice.login(“wrong”,”wrong”)
correct password;SDevice.login(“login”,”password”)
lock button;SDevice.lockButton()
long lock button;SDevice.longLockButton()

transicao.csv:
states/events,home,wrong password,correct password,lock button,long lock button
OFF,LOCKED,,,,
LOCKED,,LOCKED,UNLOCKED,,OFF
UNLOCKED,,,,LOCKED,OFF

A saída, impressa no console, acompanha a própria implementação do sistema, conforme descrito na seção “Implementação”, e foi a seguinte:

Eventos:
[home, wrong password, correct password, lock button, long lock button]

Estados:
[OFF, LOCKED, UNLOCKED]

Árvore de Transição:
OFF_0,LOCKED_0
LOCKED_0,LOCKED_1

LOCKED_0,UNLOCKED_0
LOCKED_0,OFF_1
UNLOCKED_0,LOCKED_2
UNLOCKED_0,OFF_2

Nós folha:
[LOCKED_1, OFF_1, LOCKED_2, OFF_2]

Caminhos:
OFF_0,LOCKED_0,LOCKED_1
OFF_0,LOCKED_0,OFF_1
OFF_0,LOCKED_0,UNLOCKED_0,LOCKED_2
OFF_0,LOCKED_0,UNLOCKED_0,OFF_2

Transições:
OFF,home,LOCKED
LOCKED,wrong password,LOCKED
LOCKED,correct password,UNLOCKED
LOCKED,long lock button,OFF
UNLOCKED,lock button,LOCKED
UNLOCKED,long lock button,OFF

Testes:

```
@Test
public void testaCaminho1(){
assertEquals("OFF",SDevice.getState());
SDevice.home();
assertEquals("LOCKED",SDevice.getState());
SDevice.login(“wrong”,”wrong”);
assertEquals("LOCKED",SDevice.getState());
}

@Test
public void testaCaminho2(){
assertEquals("OFF",SDevice.getState());
SDevice.home();
assertEquals("LOCKED",SDevice.getState());
SDevice.longLockButton();
assertEquals("OFF",SDevice.getState());
}

@Test
public void testaCaminho3(){
```

```
assertEquals("OFF",SDevice.getState());
SDevice.home();
assertEquals("LOCKED",SDevice.getState());
SDevice.login(“login”,”password”);
assertEquals("UNLOCKED",SDevice.getState());
SDevice.lockButton();
assertEquals("LOCKED",SDevice.getState());
}

@Test
public void testaCaminho4(){
assertEquals("OFF",SDevice.getState());
SDevice.home();
assertEquals("LOCKED",SDevice.getState());
SDevice.login(“login”,”password”);
assertEquals("UNLOCKED",SDevice.getState());
SDevice.longLockButton();
assertEquals("OFF",SDevice.getState());
}

Process finished with exit code 0
```

Neste projeto, não foi implementado a tempo a geração de testes para os caminhos ocultos da máquina de estados.

O projeto segue anexado no pacote de entrega, com os arquivos de entrada conforme descritos, portanto a sua execução irá resultar na exata saída descrita acima.